Spin the bottle game

Diego Alejandro Gómez Pardo

Laboratory in Humanoid Robotics

Humanoid Robots Group

Computer Siences Department 4

Rheinische Friedrich Wilhelms Universität Bonn

2015

Software project management

Software design and implementation description

Software project management

1. INTRODUCTION

1. Project Overview

Spin the Bottle is a party game in which several players sit/stand/kneel in a circle. A bottle is placed on the floor in the center of a circle and a player spins the bottle. There are several variations of this game, which differ depending on the age of the players [1].

Basically, the mechanic of the game of Spin the bottle with NAO compared with the original one will not have many differentiations other than instead of a circle, the people will sit in half a circle everyone looking at the robot. The first person who spins the bottle will be the one who is in front of nao and the next person who will repeat the process is the one who nao points based on the bottle pointing.

In the default presentation of the game, a maximum of 5 people can play the game, however this configuration can be changed according to the number of people. They can play standing or sitting in the floor or a chair. The bottle will be able to point into 5 different regions where the robot will look (5 regions for 5 people). If someone is in that pointed region, Nao will ask to the person to spin the bottle, otherwise he will ask to anyone to re-spin it. An example of the initial environment will be shown in the next image.

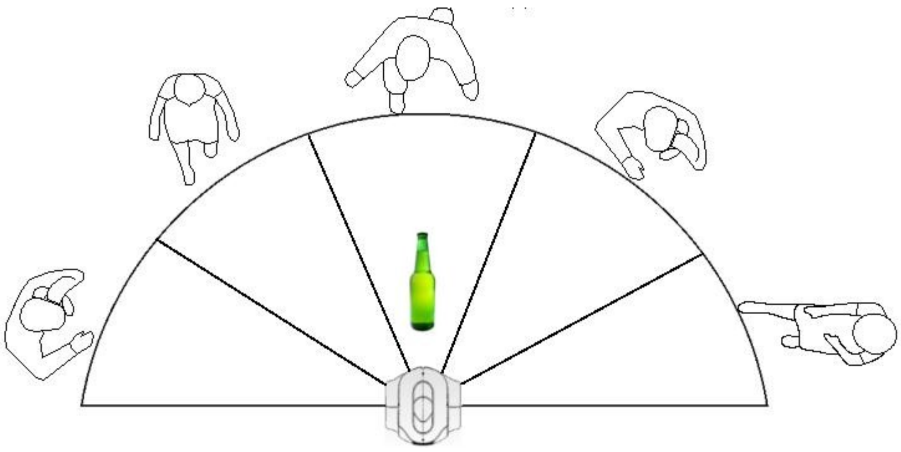

Figure 1: Spin the bottle game configuration.

2. Project Deliverables

This project is divided into 5 macro deliverables which represent in a glance, the structure of how the game must be structured. These deliverables are:

- <u>Bottle detection and pointing</u>: The implementation to detect if there is a bottle in the range of sight of the robot and also, the calculation with computer vision about where bottle is pointing in the screen.

- <u>Movement detection</u>: The code which will detect if the bottle is either spinning or not.

- <u>Pixel to World coordinates transformations</u>: A set of algorithms to calculate how much NAO has to turn its head in yaw and pitch angles in order to point with its head where the bottle does.

- People detection: The implementation to make NAO detect if a person is in its available range of sight based on face detection.
- NAO speaking messages: An implementation of messages which NAO should tell depending on game situations where it is necessary to provide instructions to users in order to maintain the normal flow of the game.

2. PROJECT MANAGEMENT PLAN

1. Bottle detection and pointing

1. Description:

   Creation of a ROS-based application able to detect first whether a bottle can completely be seen in the range of sight of the camera of the robot. After the bottle is recognized as inside the range of sight, the program will calculate with computer vision functions, the center of the bottle and the angle where it is pointing. Finally a line of 20 pixels will be sent in order to validate the calculations.

2. Deliverables.

   - Bottle detection function.
   - Angle and center of the bottle function.

3. Dependencies and constraints

   - The angle and center of the bottle can be calculated and drawn just if there is a bottle detected.

4. Risks and contingencies.

   - Light conditions (artificial and natural light) could affect the correct measurements of bottle pointing. The algorithm of calculation of angle must be enough robust as to cope with these scenarios.

2. Movement detection

1. Description

   Integration with the previous task of a function able to detect movement, mainly whether the bottle is spinning or not. Based on this one, the calculations of angle and center of the bottle can be done.

2. Deliverables and Milestones

   - Movement detection function.

3. Dependencies and constraints

- Before checking if a bottle is moving, it is necessary to check whether there is a bottle or not.

4. Risks and contingencies

- Movement detection can be false also if there is no bottle in the range of sight, so if there is no bottle in the middle, this algorithm will not be calculated.

3. Pixel to World coordinates transformations

1. Description

Incorporation of a physical movement in the neck of the robot which consists in pitch and yaw angles used to turn where the bottle points after having been rotating.

2. Deliverables and Milestones

- Pitch rotation function in Nao neck.
- Yaw rotation function in Nao neck.

3. Dependencies and constraints

- In order to NAO rotate in pitch and yaw, it is necessary first to have calculated a pointing angle.
- NAO cannot turn its neck in angles between 225° and 315°, because of this the game is used in a semi circle.

4. People detection

1. Description

Detection of people in the range of sight of the robot camera based on the detection of eyes and shape of a person's face.

2. Deliverables and Milestones

- based-camera face detection function.

3. Dependencies and constraints

- To detect faces it is necessary to count with a person whose eyes are open and has his face oriented towards the robot camera.

4. Risks and contingencies

- Due to the low quality resolution of the camera used in order to gain performance, it is possible that some faces are not recognized correctly. In order to mitigate the false negative results, the resolution of the camera is increased to twice of original in order to detect more faces.

5. NAO speaking messages

1. Description

Additions of NAO speaking messages in cases where a user's action in the middle of the game is necessary such as when the bottle is out of the range of sight of the robot, when the bottle points to a place where NAO cannot look or when a person is asked to spin the bottle.

2. Deliverables and Milestones

- A class containing all of the messages used to maintain the normal flow of the game.

3. Dependencies and constraints

- an out of the normal flow action must be fired in order to get these messages.

6. Timetable of tasks

Task \ Date	February	March	April	May	June	July
Bottle detection		🟨				
Bottle pointing	🟨	🟨	🟨	🟨		
Movement detection		🟨		🟨		
Pixel to world coordinates					🟨	
People detection						🟨
Speaking messages						🟨

Software design and implementation description

3. SOFTWARE DESIGN AND IMPLEMENTATION

1. Brief general software design

The design of this software is conceived in the object oriented paradigm in order to provide low coupling and high cohesion to allow improvements in the program. The application is divided in the next classes which will be explained in the following table:

Class	Definition
My_subscriber	Contains the subscriber node which subscribes and publishes to the topics provided by NAO for this game (cameras from Naoqi_sensors, pose_controller from Nao_pose, and nao_say from Nao_speech respectively) additionally executes the game (bottle detection, movement detection, pointing angle, world coordinates, recognize faces)
PreGame	Contains algorithms to determine whether there is a bottle in the range of sight and also whether there is movement detected
LineProjectionE	Contains functions to define the pointing place of the bottle (pointing angle, and center of it) and the edge of the screen towards it is pointing.
WorldCoordinates	Contains algorithms to convert the angle where the bottle points in pixel coordinates to the angle in yaw and pitch where the robot should turn in world coordinates
DetectFace	Contains a function to detect human faces based on its eyes and face shape.
ActionClientSay	A class which contains a Node to communicate to the Nao_speech package.
ActionNaoNeck	A class which contains a Node to communicate to the Nao_pose package.

In order to cope with the problems of bottle pointing, changing from pixel to world coordinates, face detection, movement and bottle detection, and the lack of communication from the robot to the human in the game, the next strategies were used:

2. Design Phase 1: Bottle detection and pointing

1. Strategy 1: Edge detection and Hough lines + K-means algorithm based orientation

1. Description

The first approach to solve the problem of bottle pointing was made using edge detection and based on this, the OpenCV function of Hough lines was applied. To go deeper on this strategy it is important to define each of the parameters of these two functions

- Canny edge detector:

Canny edge detector is an algortihm to detect edges in an image (high pass filter). developed by John F. Canny in 1986. Canny algorithm aims to satisfy three main criteria:

- Low error rate: Meaning a good detection of only existent edges.
- Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.
- Minimal response: Only one detector response per edge [2].

The opencv Function for canny Edge detector is the following one:

Canny(image_src, image_dest, lowThreshold, HighThreshold, kernel_size);

Where the parameters have the following meaning:

- image_src - Source image, grayscale .
- image_dest - Output of the detector.
- LowThreshold - Lower threshold accepted in order to do the edge detection.
- HighThreshold - Set in the program as three times the lower threshold (following Canny's recommendation) .
- kernel_size - The size of the Sobel kernel to be used internally.

An example of the result of Canny edge detector with the bottle is this

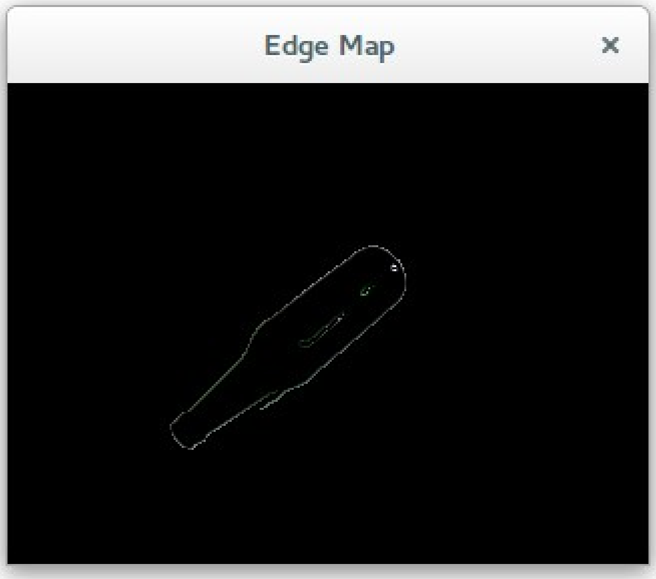

Figure 2 Canny Edge detector applied in image

- Probabilistic Hough Lines Transform

This function Finds line segments in a binary image using the probabilistic Hough transform [3]. The idea under this function was to detect the two biggest parallel lines which could help to find the central point of the bottle and to draw a middle line between those largest edges .

The opencv function for probabilistic hough lines is

HoughLinesP(InputArray image, OutputArray lines, double rho, double theta, int threshold, double minLineLength=0, double maxLineGap=0)

where:

- image - 8-bit, single-channel binary source image.
- Lines - Output vector of lines. Each line is represented by a 4-element vector which represent the starting and final point in x,y of a line.
- Rho - Distance resolution of the accumulator in pixels.
- theta - Angle resolution of the accumulator in radians.
- threshold - Only those lines bigger than the threshold are returned.
- minLineLength - Minimum line length. Line segments shorter than that are rejected.
- maxLineGap - Maximum allowed gap between points on the same line to link them.

An example of the result of Probabilistic Hough Lines with the bottle is this

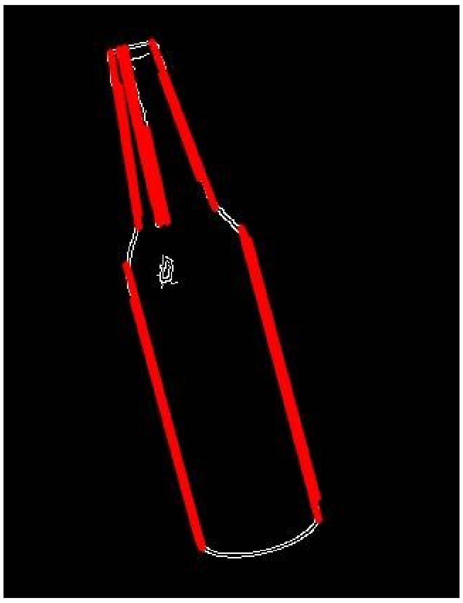

Figure 3 Probabilistic Hough Lines after and Edge detector applied in image

- K-means clustering algorithm:

  In order to extend the line where the bottle is pointing, the k-means algorithm was used in order to divide the image in 2 clusters of images. After using this one, 2 squares were used to divide the bottle in 2 and the area with less foreground pixels is would be the one where the bottle should point.

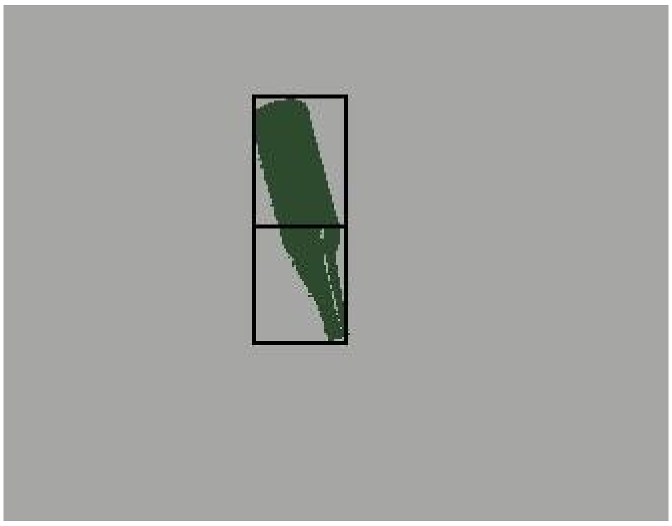

Figure 4 Orientation edge where the bottle points using k-means clustering.

The Opencv function for the k-means clustering is this one:

kmeans(InputArray data, int K, InputOutputArray bestLabels, TermCriteria criteria, int attempts, int flags, OutputArray centers=noArray())

where:

- data – source image.
- K – Number of clusters to split the set by.
- Bestlabels – Input/output integer array that stores the cluster indices for every sample.
- criteria – The algorithm termination criteria, that is, the maximum number of iterations and/or the desired accuracy.
- attempts – Number of times the algorithm is executed using different initial labellings.
- Flags – used KMEANS_RANDOM_CENTERS to select random initial centers in each attempt
- centers – Output matrix of the cluster centers, one row per each cluster center [4]

2. Drawback on this strategy

Since photos were taken with a cellphone in a high resolution, this allowed the detection of edges easily. However in the low camera resolution used by NAO, most of the edges, or even none of them which appeared in the phone photograph, reappeared in the NAO camera video, because of this, it was necessary to change of strategy. This happened because of the discontinuities of the lines, additionally in some cases the algorithm did not work properly because the bottle longest lines from the high resolution photo sometimes were cut in smaller segments and the result of the line projection was wrong. Orientation did not present any problem and was kept for the next step.

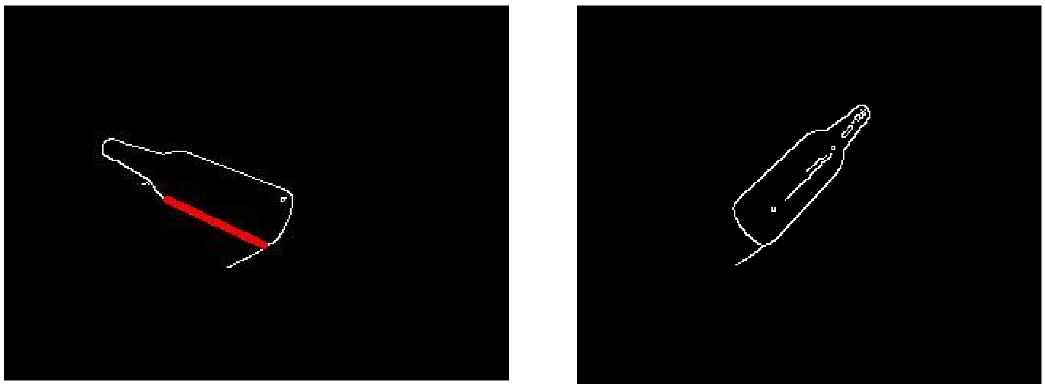

Figure 5 Result of applying Hough lines wth Nao cameras

2. Strategy 2: PCA algorithm + K-means clustering orientation

   1. Description

Since the previous strategy on orientation went wrong, the next step was to change from edge detection to segmentation, and this is why the functions of threshold previous color change was adapted.

- Convert color function

  This function converts the channel of an image into another one (type or number of channels) it will be used here just to convert from an image color to a grayscale one.

  cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)

  where:

  - src – input image.
  - dst – output image.
  - code – color space conversion code (the code used was CV_GRAY2BGR).
  - dstCn – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from src and code.

- Threshold function

  Threshold is the simplest segmentation method , it separates regions of a range corresponding to a variation of intensity between foreground and background pixels.

  To differentiate the pixels of interest from the rest (which will eventually be rejected), a comparison of each pixel intensity value with respect to a threshold is performed. The function in openCV is the next one

  threshold(InputArray src, OutputArray dst, double thresh, double maxval,

int type)

where:

- src – input array , dst – output array of the same size as src.
- thresh – threshold value.
- maxval – maximum value to asign to pixels above the used threshold
- type – thresholding type, THRESH_BINARY was the type used.

An example of the result after using threshold function is this one:

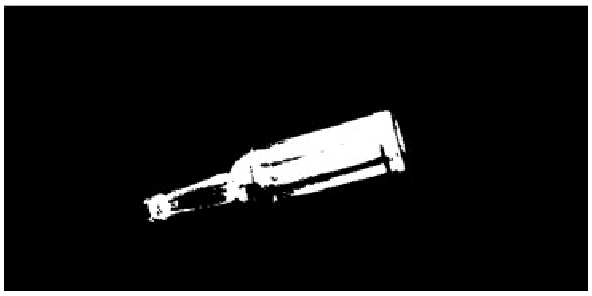

Figure 6 Image with threshold function applied

- PCA Algorithm

Principal Component Analysis (PCA) is a statistical procedure that extracts the most important features of a dataset. In this case what it does is to find the direction along which data in an image varies the most the result. Running PCA on the set of points in a 2d diagram consist of 2 vectors called eigenvectors which are the principal components of the data set. [5]

Applying PCA algorithm in the example bottle produced the eigenvectors as following.

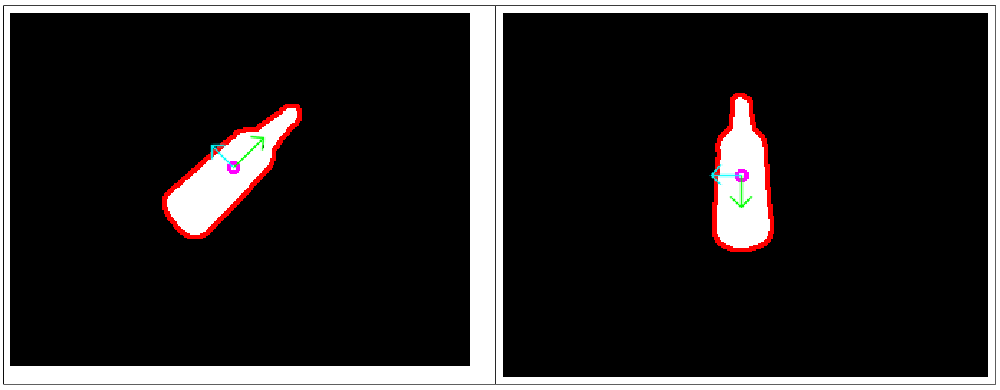

Figure 7 PCA Algorithm applied in bottle in low bright conditions

For this case, the only eigenvector needed was the one following the longest set of pixel data

2. Drawback on this strategy

In dark days conditions where sunlight is very weak (no strong shadow formation) the algorithm works very well, however when light has bright conditions or artificial light is turned near the bottle on, not just the shadows of the bottle produce angle projection problems, but also the orientation algorithm does not work correctly. Some examples of this are the next ones.

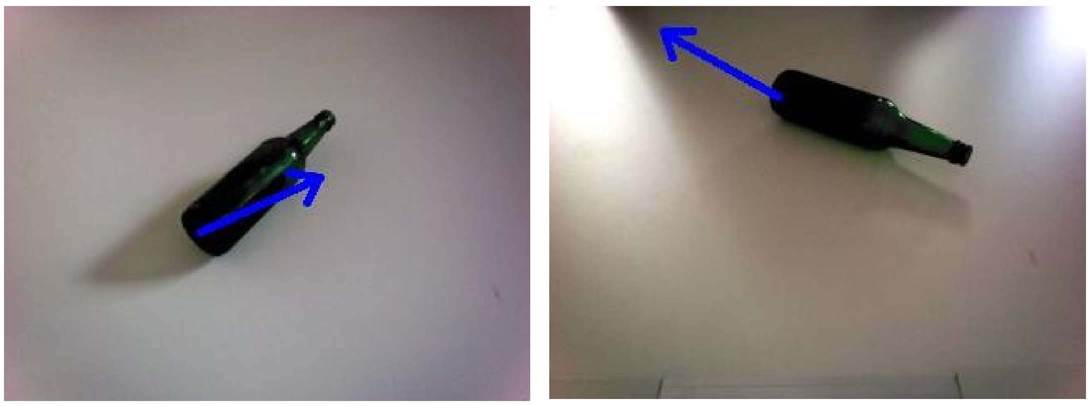

Figure 8 PCA Algorithm applied in highly bright conditions.

Because of this it was decided that PCA algorithm under these preprocessing conditions would not be effective. The orientation of angle was also rejected because of shadow conditions.

3. Strategy 3: single green channel + fit ellipse function + convex hull + FillConvexPoly

   1. Description

      - Single green channel

        The main problem of light reflection in glass objects when doing an OpenCV image processing is that in very bright levels, the reflected color can even reach pixel color levels with same magnitude as the ones in the bottle. A way humans cope with this problem is checking the depth of the image and automatically discarding the shadow as part of the bottle, something that cannot be done with a 2d camera since it in the process of converting 3d pixels to 2d ones, discards the depth of the reflected area.

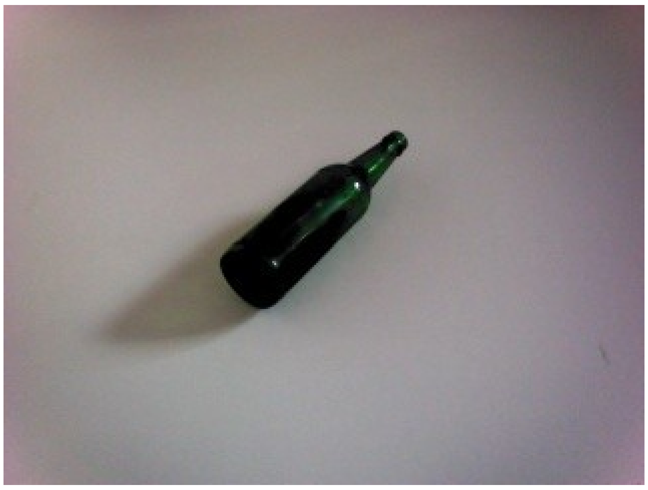

Figure 9 PCA bottle color reflection from bright natural and aritificial light

In the image above it can be seen that the shadow of the bottle is presenting a green color in the contour of the bottle, however this contour is not complete since part of that contour is occluded by the bottle, moreover the range of positions which the bottle could have makes this problem non-trivial. This problem affects projection as well as projection of the bottle.

Based on observations made on the bottle contour pixels, it was found that some parts of it and just it have pixels with value 0 (shadow or reflection present 0 values of completely dark pixels). Based on this, the shadow reflection is rejected

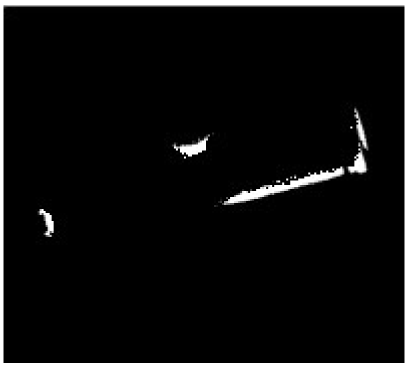

Figure 10 PCA Result of applying the values of single green channel on the picture.

- FitEllipse function

Once the algortihm of zero value pixels in green channel was done, this algorithm was tested on the PCA algorithm. The result of the combinations of both algorithms could not be satisfactory since PCA follows the principal components of the data set and in this case the pointing would be from the bottom of the bottle but not on the center of it. This would cause more inexact measurements.

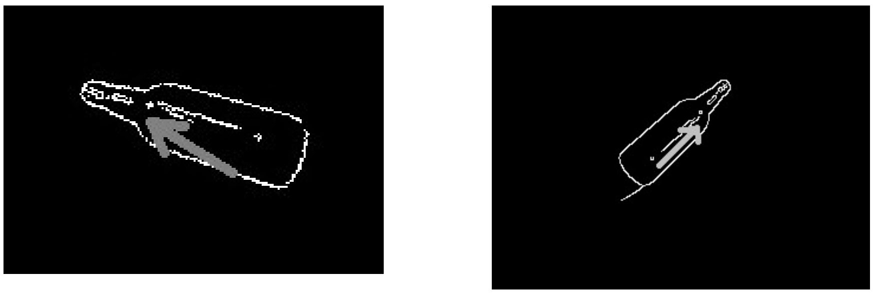

Figure 10 PCA algorithm applied on zero value pixels in green channel

Due to these inconvenients a new function was used which name is FitEllipse, and additional to this, the convex hull function was added. Basically the idea of this final step was to get all of the contours obtained by the green value pixels, merge all of them in the function ConvexHull and then apply the fitEllipse functions

The ConvexHull function finds the convex hull of a 2D point set using the Sklansky's algorithm [6]

convexHull(InputArray points, OutputArray hull, bool clockwise=false, bool returnPoints=true)

where

- points – Input 2D point set, stored in std::vector or Mat.
- hull – Output convex hull. It is either an integer vector of indices or vector of points.
- clockwise – Orientation flag. If it is true, the output convex hull is oriented clockwise.
- returnPoints – Operation flag. In case of a matrix, when the flag is true, the function returns convex hull points. Otherwise, it returns indices of the convex hull points. [7]

The fitEllipse function just draws an elipse based on a set of points. The final result will explain the function better

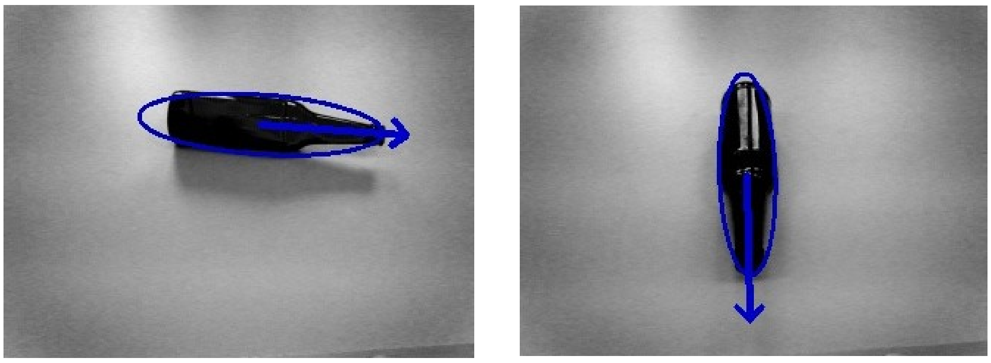

Figure 11 Fit ellipse function applied on zero value pixels in green channel

- Filled contours: The algorithm of zero values on green channel and the OpenCV convexhull function were also used in order to find the orientation of the bottle in a better way. They were combined with an openCV function called FillConvexPoly, which additionally removed every brightness defect allowing to identify completely the orientation of the bottle. The results above show the final result of applying both orientation and projection.

3. **Design Phase 2:** Movement detection

   1. Description: Nao_movement_detect vs Difference of thresholds in OpenCV.

      The selection of the difference of thresholds in openCV was given since the NAO node of movement detection did not register any change when the bottle was spinning. This test was made using just Ros Echo function. Because of this an OpenCV function to calculate difference of thresholds between 2 frames (current and previous) was used.

      The main idea is to calculate a threshold based on the green pixel channels value (being 0 the only value taken into account) in each of the frames and after that a subtraction between the 2 frames is made and the result of this value is converted to absolute. Detecting no movement would result in a frame completely full of zeros, however due to the NAO camera noise sensor, a threshold of 50 pixels with values different to zeros was allowed in order to make the program not to say all the time that there was movement.

2. Drawback on this strategy

   Due to the noise and since it appears with a gaussian probability, there is not a perfect way or pixel quantity to define absolute movement or not, what implies that there will be some false positives hence they affect the efficiency of the program, however not the effectiveness.

4. **Design Phase 3**: Pixel to World coordinates transformations

   1. Description: Inverse of intrinsics matrix -vs- Mean of mapping pixel values – world

      The choose of Mean of mapping pixel values was decided just in terms of efficiency -vs- effectiveness. Choosing the inverse of intrinsics matrix would have allowed Nao to look exactly to the point where the bottle points (just to cope with the noise of the camera), however since it was decided with the supervisor to separate the space in ranges, then some decimals or a couple of degrees would not make too much change, this is why the Mean of Mapping pixel values was chosen

      The idea behind this algorithm is that normally the center of the bottle is between the middle of the field of view of the nao and 10 pixels counting from the Nao_Bottom camera and 25 cms counting from the floor. Based on this a mapping between the maximum and minimum coordinates which nao can use to see the complete bottle was used in this algorithm.

   2. Drawback on this strategy

      As it was said before, probably it will not provide exact, but at least faster results (avoiding Matrix multiplication for just a set of simple multiplications). Probably in order to play with more people, a precise calculation is necessary.

5. **Design Phase 4**: People detection

   1. Description: Nao_face_detect -vs - Haar cascades

      Nao has a special feature in his face_detect node and it is that it can recognize a face and all of the features on it (nose, lips, eyebrows, eyes, etc), however it cannot say neither the number of faces, nor exactly where the face is located. Due to this inconvenient, the option of OpenCV Haar Cascades was selected

      Haar cascades is an approach which uses machine learning, where a cascade function is trained based on a set of positive and negative images in this case representing a face. After the function is trained, a set of features can be extracted from it (normally eyes) all of the features can be easily calculated based on integral images calculations around the person's face

   2. Drawback on this strategy

The main drawback of this strategy is the fact that people must be looking at the robot and also that they must have their eyes open when the robot makes the face detection, otherwise it will yield a false negative answer, asking for any person to spin the bottle but the one who the bottle pointed at.

6. **Miscellaneous and additional design**: Nao_speak + Nao_pose

These 2 nodes correspond to some of the most basic functions which NAO counts with. From Nao_pose the topic pose_controller is used in order to move the robot neck. This procedure is done in 2 different situations. The first one is when NAO looks at the bottle in order to check where it points, the second one is when NAO turns its pitch and yaw angles in order to look where the bottle points. The second topic corresponds to nao_say from the package Nao_speech. For this one, a message with a string is sent to the robot via this topic in order to make it say what is specified. Thise functions were used

References

[1] Brian Sutton-Smith, B. G. Rosenberg. "Sixty Years of Historical Change in the Game Preferences of American Children", The Journal of American Folklore. Volume 74, Number 291, January – March 1961. pages 17-046. ISSN 00218715

[2] http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

[3] http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html

[4] http://docs.opencv.org/modules/core/doc/clustering.html

[5] http://docs.opencv.org/ref/master/d1/dee/tutorial_introduction_to_pca.html

[6] Sklansky, J., Finding the Convex Hull of a Simple Polygon. PRL 1 $number, pp 79-83 (1982)

[7] http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=convexhull#sklansky82